International Journal of Technological Exploration and Learning (IJTEL)
ISSN: 2319-2135
Volume 2 Issue 5 (October 2013)

An Efficient FPGA Implementation of Binary Coded Decimal Digit Adders and Multipliers

G. Shaik Baba Junaid
Research Scholar

Y.Babasalauddin
Assistant Professor

Abstract---**Decimal arithmetic has gained high impact on the overall performance of today's financial and commercial applications. Decimal arithmetic is pervasive in human-oriented applications but has a limited use in numerical data processing. Decimal additions and multiplication plays a main role in decimal operations which is used in any decimal arithmetic algorithm. Decimal digit adders and decimal digit multipliers are usually the building blocks for higher order decimal adders and multipliers. FPGAs provide an efficient hardware platform that can be employed for accelerating decimal algorithms. In this paper, different designs for two decimal digit adders and one decimal digit multiplier are proposed. The proposed designs were described, functionally tested, and implemented using VHDL and the Xilinx ISE 10.1 targeting Xilinx Vertix-5 XC5VLX30-3 FPGA. This design is mainly used to decrease the area and as well as increase the functioning speed. Implementation results and comparison with existing designs are provided.**

Keywords-FGPA; Binary Coder; Digit Adder; Multiplier.

I. INTRODUCTION

In electronics, an adder or summer is a digital circuit that performs addition of numbers. In many computers and other kinds of processors, adders are used not only in the arithmetic logic unit(s), but also in other parts of the processor, where they are used to calculate addresses, table indices, and similar operations .Although adders can be constructed for many numerical representations, such as binary-coded decimal or excess-3, the most common adders operate on binary numbers. In cases where two's complement or ones' complement is being used to represent negative numbers, it is trivial to modify an adder into an adder–subtractor. Other signed number representations require a more complex adder. The half adder adds two single binary digits A and B. It has two outputs, sum (S) and carry (C). The carry signal represents an overflow into the next digit of a multi-digit addition. The value of the sum is 2C + S. The simplest half-adder design, pictured on the right, incorporates an XOR gate for S and an AND gate for C. With the addition of an OR gate to combine their carry outputs, two half adders can be combined to make a full adder.

A full adder adds binary numbers and accounts for values carried in as well as out. A one-bit full adder adds three one-bit numbers, often written as A, B, and C_{in}; A and B are the operands, and C_{in} is a bit carried in from the next less significant stage.[2] The full-adder is usually a component in a cascade of adders, which add 8, 16, 32, etc. binary numbers. The circuit produces two-bit output, output carry and sum typically represented by the signals C_{out} and S, Carry-lookahead adder to reduce the computation time, engineers devised faster ways to add two binary numbers by using carry-lookahead adders. They work by creating two signals (P and G) for each bit position, based on whether a carry is propagated through from a less significant bit position (at least one input is a '1'), generated in that bit position (both inputs are '1'), or killed in that bit position (both inputs are '0'). In most cases, P is simply the sum output of a half-adder and G is the carry output of the same adder. After P and G are generated the carries for every bit position are created. Some advanced carry-lookahead architectures are the Manchester carry chain, Brent–Kung adder, and the Kogge–Stone adder.

II. RELATED WORK

J. M. Rabaey proposed on Digital Integrated Circuits," IEEE Trans. on VLSI Systems, 2003. we proposed an area-efficient carry select adder by sharing the common Boolean logic term. After logic simplification and sharing partial circuit, we only need one XOR gate and one inverter gate in each summation operation as well as one AND gate and one inverter gate in each carry-out operation.

O. Bedrij proposed on Carry Select Adder A large, extremely fast digital adder with sum selection and multiple-radix carry is described. Boolean expressions for the operation are included. The amount of hardware and the logical delay for a 100-bit ripple-carry adder and a carry-select adder are compared. The adder system described increases the speed of the addition process by reducing the carry-propagation time to the minimum commensurate with economical circuit design.

III. FIELD PROGRAMMABLE GATE ARRAYS

Before the advent of programmable logic, custom logic circuits were built at the board level using standard components, or at the gate level in expensive application-specific (custom) integrated circuits. The FPGA is an integrated circuit that contains many (64 to over 10,000) identical logic cells that can be viewed as standard components. Each logic cell can independently take on any one of a limited set of personalities. The individual cells are interconnected by a matrix of wires and programmable switches. A user's design is implemented by specifying the simple logic function for each cell and selectively closing the switches in the interconnect matrix. The array of logic cells and interconnects form a fabric of basic building blocks for logic circuits. Complex designs are created by combining these basic blocks to create the desired circuit.

A field-programmable gate array (FPGA**)** is an integrated circuit designed to be configured by the customer or designer after manufacturing hence "field-programmable". The FPGA configuration is generally specified using a hardware description language (HDL), similar to that used for an application-specific integrated circuit (ASIC). FPGAs can be used to implement any logical function that an ASIC could perform. The ability to update the functionality after shipping, partial re-configuration of the portion of the design and the low non-recurring engineering costs relative to an ASIC design (not withstanding the generally higher unit cost), offer advantages for many applications.

IV. FPGA Architecture for the Challenge

The FPGA is an array or island-style FPGA. It consists of an array of logic blocks and routing channels. Two I/O pads fit into the height of one row or the width of one column, as shown Fig 1. All the routing channels have the same width (number of wires). Each circuit must be mapped into the smallest square FPGA that can accommodate it. For example, a circuit containing 14 logic blocks and 10 I/O pads would be mapped into an FPGA consisting of a 4x4 array of logic blocks. Note that three of the twenty benchmark circuits used in the FPGA challenge (big key, des, and dsip) are pad-limited in this FPGA architecture.

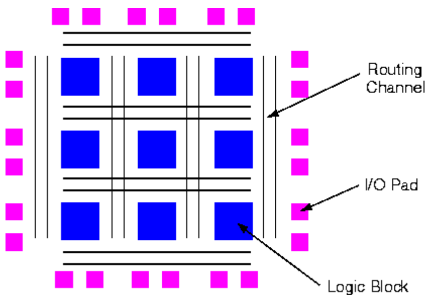


Figure 1. FPGA structure

The FPGA logic block consists of a 4-input look-up table (LUT), and a flip flop, as shown in Fig 2. There is only one output, which can be either the registered or the unregistered LUT output. The logic block has four inputs for the LUT and a clock input. Since the clock is normally routed via a special-purpose dedicated routing network in commercial FPGAs. That is, you can completely ignore the clock net, since it is assumed to be routed on a special global network.

A. Logic Block Strucuture

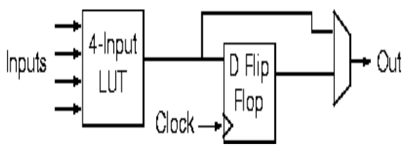


Figure 2. Logic Block Structure

Each logic block input pin can connect to any one of the wiring segments in the channel adjacent to it. Each logic block output pin can connect to any of the wiring segments in the channels adjacent to it. (In the usual FPGA terminology, then, Fc = the number of tracks per channel, W),as shown in Fig.3

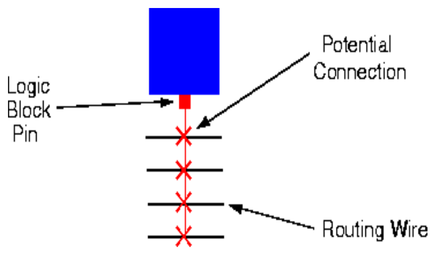


Figure 3. Logic Block Pin to Routing Channel Interconnect

Similarly, an I/O pad can connect to any one of the wiring segments in the channel adjacent to it. For example, an I/O pad at the top of the chip can connect to any of the W wires (where W is the channel width) in the horizontal channel immediately below it. Whenever a vertical and a horizontal channel intersect there is a switch box. In this architecture, when a wire enters a switch box, there are three programmable switches that allow it to connect to three other wires in adjacent channel segments. The pattern, or topology, of switches used in this architecture is the planar or domain-based switch box topology. In this switch box topology, a wire in track number one connects only to wires in track number one in adjacent channel segments, wires in track number 2 connect only to other wires in track number 2 and so on. The Fig. 4 illustrates the connections in a switch box.

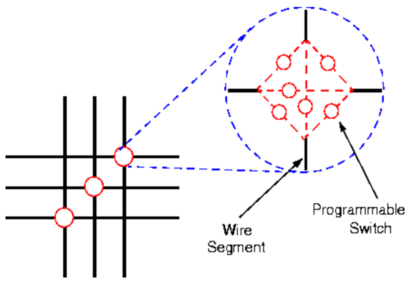


Figure 4. Switch Box Topology

Much of the prior research on FPGA routing architectures has assumed that input pin doglegs are possible. An input pin dogleg occurs when one connects more than one routing wire segment to the same logic block input pin. The input pin dogleg then, allows one to change tracks by connecting two wiring segments to a logic block input pin. Input pin doglegs are not allowed in the FPGA architecture to be used in the challenge; i.e. only one wire segment may be connected to a logic block input pin.

However, the routing architecture above is not the routing architecture that is actually implemented by SRAM-based FPGAs. As shown below, commercial SRAM-based FPGAs normally place a buffer between routing tracks and the input pins to which they can connect to enhance speed. As well, to save area, the connection from routing wire segment to input pin is made via a multiplexer, not via a set of independent pass transistors. Accordingly, it is not possible to connect two wire segments together via an input pin, and input pin doglegs are not possible.

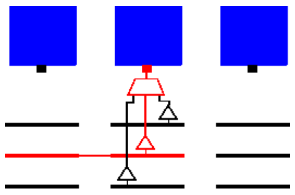


Figure 5. More Realistic Architecture

Note that it is possible to connect a logic block output pin to multiple wire segments in commercial FPGAs. Accordingly, such connections are allowed in the FPGA challenge. Applications of FPGAs include digital signal processing, software-defined radio, aerospace and defense systems, ASIC prototyping, medical imaging, computer vision, speech recognition, cryptography, bioinformatics, computer hardware emulation, radio astronomy, metal detection and a growing range of other areas.

V. RESULTS

A. SIMULATION RESULTS OF EXISTING SYSTEM

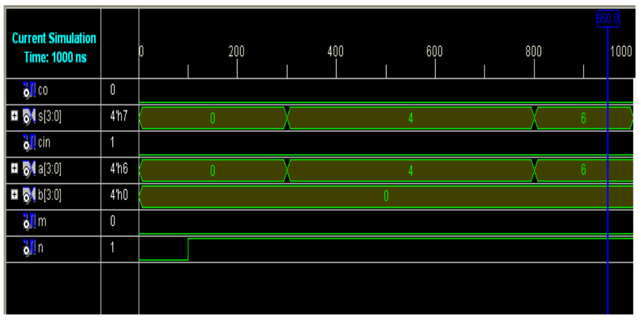
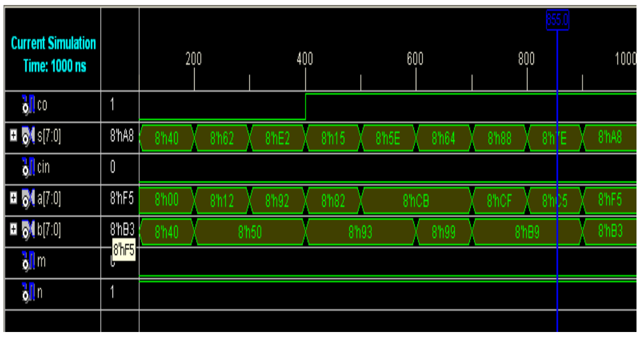
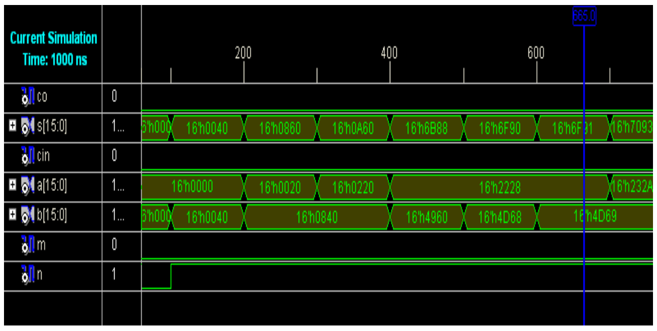
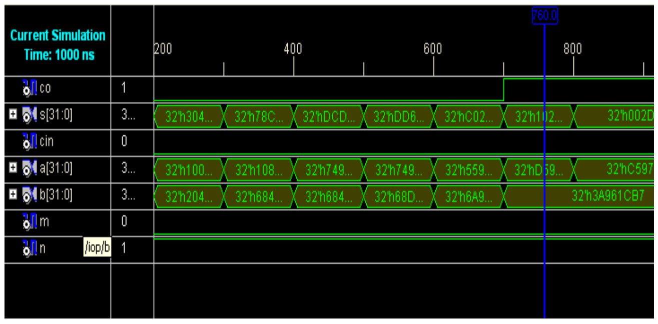


Figure 6. Simulation Result


Current Simulation Time: 1000 ns
co 1
s[7:0] 8'hA8 8'h40 8'h62 8'hE2 8'h15 8'h5E 8'h64 8'h88 8'hAE 8'hA8
cin 0
a[7:0] 8'hF5 8'h00 8'h12 8'h92 8'h82 8'hCB 8'hCF 8'hD5 8'hF5
b[7:0] 8'hB3 8'h40 8'h50 8'h93 8'h99 8'hB9 8'hB3
m 0
n 1
Figure 7. Simulation Result


Current Simulation Time: 1000 ns
co 0
s[15:0] 1... 16'h0040 16'h0860 16'h0A60 16'h6B88 16'h6F90 16'h6F91 16'h7093
cin 0
a[15:0] 1... 16'h0000 16'h0020 16'h0220 16'h2228 16'h232A
b[15:0] 1... 16'h0040 16'h0840 16'h4960 16'h4D68 16'h4D69
m 0
n 1
Figure 8. Simulation Result


Current Simulation Time: 1000 ns
co 1
s[31:0] 3... 32'h304... 32'h78C... 32'hDCD... 32'hDD6... 32'hC02... 32'h1112... 32'h002D
cin 0
a[31:0] 3... 32'h100... 32'h108... 32'h749... 32'h749... 32'h559... 32'hD59... 32'hC597
b[31:0] 3... 32'h204... 32'h684... 32'h684... 32'h68D... 32'h6A9... 32'h3A961CB7
m 0
n 1
Figure 9. Simulation Result

B. SIMULATION RESULTS OF PROPOSED DESIGN

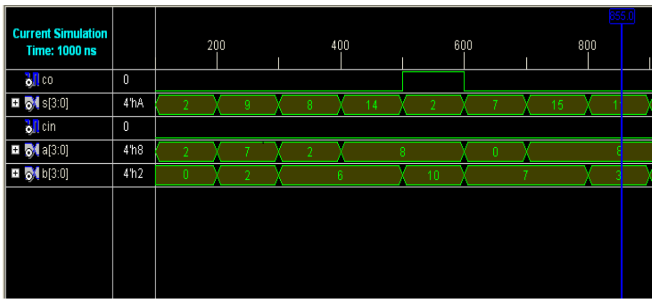
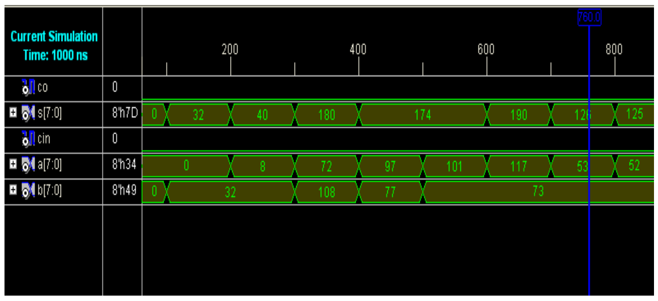
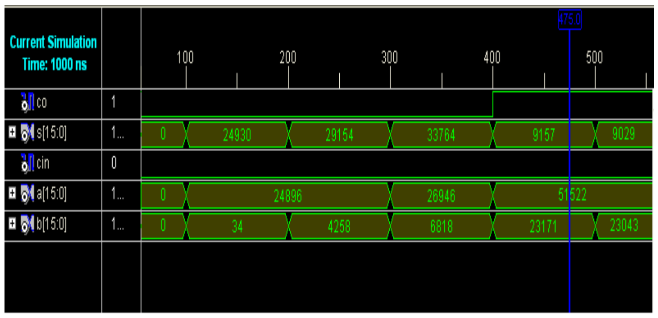
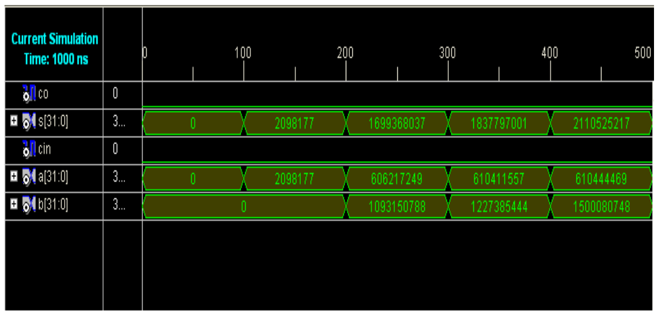


Figure 10. Simulation Result


Current Simulation Time: 1000 ns
co 0
s[7:0] 8'h7D 0 32 40 180 174 190 126 125
cin 0
a[7:0] 8'h34 0 8 72 97 101 117 53 52
b[7:0] 8'h49 0 32 108 77 73
Figure 11. Simulation Result


Current Simulation Time: 1000 ns
co 1
s[15:0] 1... 0 24930 29154 33764 9157 9029
cin 0
a[15:0] 1... 0 24896 26946 51522
b[15:0] 1... 0 34 4258 6818 23171 23043
Figure 12. Simulation Result


Current Simulation Time: 1000 ns
co 0
s[31:0] 3... 0 2098177 1699368037 1837797001 2110525217
cin 0
a[31:0] 3... 0 2098177 606217249 610411557 610444469
b[31:0] 3... 0 1093150788 1227385444 1500080748
Figure 13. Simulation Result

VI. COMPARISON RESULTS

To compare the circuit performance with three different architectures, 32-bit carry ripple adder, 32-bit carry select adder, and 32-bit area-efficient carry select adder that is proposed in this paper. As for the transistor count, the transistor count of our proposed area-efficient carry select adder could be reduced to be very close to that of carry ripple adder; however, the transistor count in the conventional carry select adder is nearly double as compared with the proposed design. This result shows that sharing common Boolean logic term could indeed achieve a superior performance in aspect of transistor count. The area-efficient carry select adder can also achieve an outstanding performance in power consumption. Power consumption can be greatly saved in our proposed area-efficient carry select adder because we only need one XOR gate and one INV gate in each summation operation as well as one AND gate and one OR gate in each carry-out operation after logic simplification and sharing partial circuit. Because of hardware sharing, we can also significantly reduce the occurring chance of glitch. Besides, the improvement of power consumption can be more obvious as the input bit number increases.

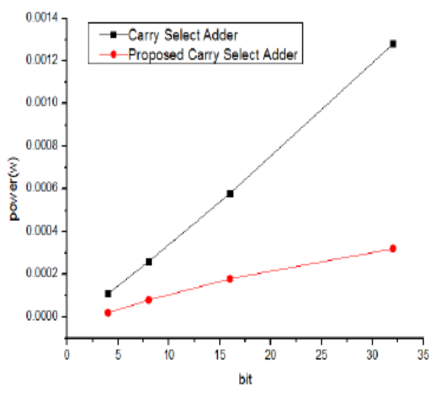


Figure 14. The simulation results of the power consumption comparison in the proposed area-efficient carry select adder and the conventional carry select adder.

We simulated the power consumption in the proposed area-efficient adder and the conventional carry select adder with 4, 8, 16, and 32-bit respectively in tsmc 0.18um CMOS technology. Figures above shows the simulation results of the proposed area-efficient carry select adder and the conventional carry select adder. The power consumption difference between these two designs is small in the case of 4-bit input word length. Since the conventional carry select adder consists of the duplicated adder cells to prepare both the possible output values for the corresponding carry input values in advance. It not only needs larger hardware area, but also generates more glitch signals because of propagation path difference. Therefore, as the input bit number increases, the slope of power consumption increase in the conventional carry select adder would be larger than that in our proposed design. As the input bit number of the conventional carry select adder increases to 32-bit, the power consumption in the conventional carry select adder will be 3.3 times larger than that in our proposed area-efficient carry select adder.

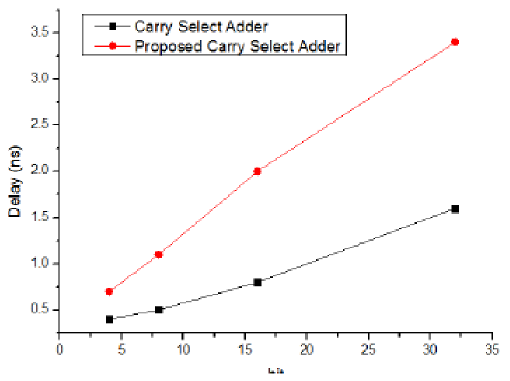


Figure 15. The simulation results of the computation speed comparison in the proposed area-efficient carry select adder and the conventional carry select adder.

VII. CONCLUSION

In this work, an area-efficient carry select adder is proposed. By sharing the common Boolean logic term, we can remove the duplicated adder cells in the conventional carry select adder. In this way, the transistor count in a 32-bitcarry select adder can be greatly reduced from 1947 to 960. Moreover, the power consumption can be reduced from 1.26mw to 0.37mw as well as power delay product reduced from 2.14mw*ns to 1.28mw*ns. By retaining part of parallel architecture of conventional carry select adder, we can still maintain some competitiveness in speed. In this way, our area-efficient adder can perform with nearly the same transistor count, nearly the same power consumption, but with faster speed and lower PDP as compared with the carry ripple adder. This design is mainly used to reduce the area and as well as to increase the functioning speed.

REFERENCES

[1] IBM Corporation, "Decimal FAQ," http://www2.hursley.ibm.com/decimal/decifaq1.html.

[2] M. F. Cowlishaw, "Decimal floating-point: Algorism for computers," in *Proceedings of the 16th IEEE Symposium on Computer Arithmetic(ARITH-16'03)*, Washington, DC, USA, 2003, ARITH '03, pp. 104–, IEEE Computer Society.

[3] A. Tsang and M. Olschanowsky, "A study of database 2 customer queries," Tech. Rep., IBM Technical Report, IBM Santa Teresa Laboratory, an Jose, CA, April 1991.

[4] M. S. Schmookler and A. Weinberger, "High speed decimal addition," *IEEE Transactions on Computers*, vol. 20, pp. 862–866, 1971.

[5] H. Wetter W. Bultmann, W. Haller and A. Worner, "Binary and decimal adder unit," 2001.

[6] J. Thompson, I. Karra, and M. J. Schulte, "A 64-bit decimal floatingpoint adder," in *Proceedings of the IEEE Computer Society Annual Symposium on VLSI*, 2004, pp. 297–298.

[7] A. V´azquez and E. Antelo, "Conditional speculative decimal addition," Nancy, France, 2006, pp. 47–57.

[8] R. H. Larson, "High speed multiply using four input carry save adder," *IBM Technical Disclosure Bulletin*, pp. 2053–2054, 1973.

[9] R. H. Larson, "Medium speed multiply," *IBM Technical Disclosure Bulletin*, p. 2055, 1973.

[10] G. Jaberipur and A. Kaivani, "Binary-coded decimal digit multipliers," *IET Computers and Digital Techniques*, vol. 1, no. 4, pp. 377–381, 2007.